EECS 427
Lecture 12: Multipliers
WH 11.9.1-11.9.3

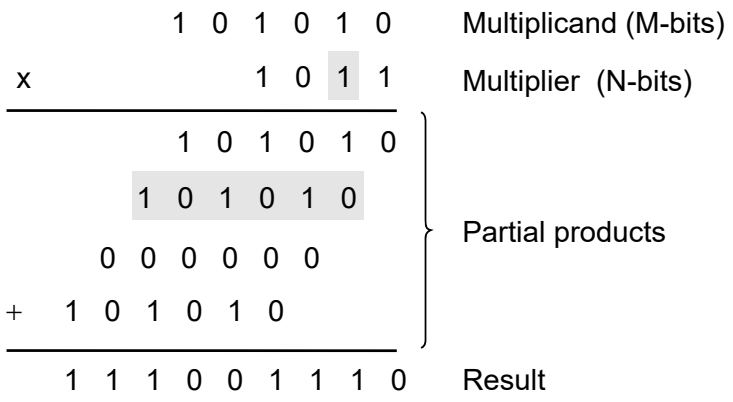
1

Lecture Overview

- Multipliers are vital in digital signal processing and general purpose processors & GPUs
- They are speed limiting – very slow operations

2

Binary Multiplication

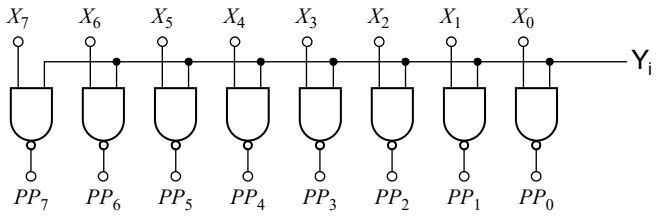


Key points

- MxN multiplication fits in N+M bits
- 2s complement multiplication is more difficult: Either convert to **+** numbers and keep track of original signs OR use Booth's algorithm
- Major steps are:
  1) Partial product generation
  2) Partial product accumulation
  3) Final addition (done using fast carry lookahead techniques)

Generating Partial Products

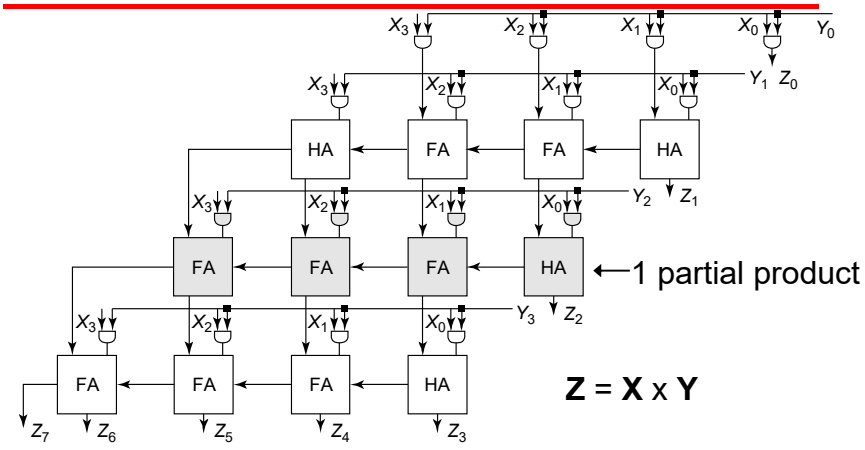
- All partial products: AND



- Booth's recoding – reduction of partial product count (more later)

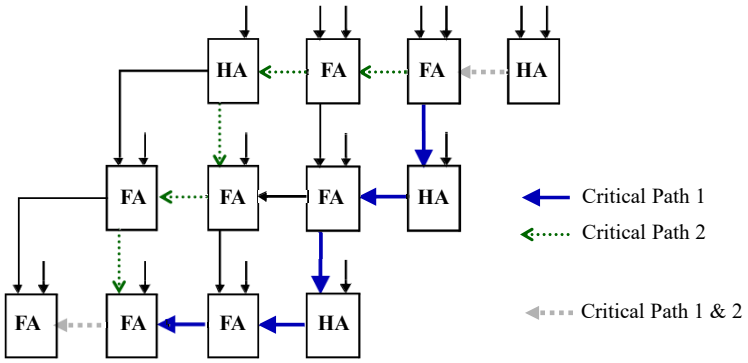
5

The Array Multiplier



6

MxN Array Multiplier — Critical Path

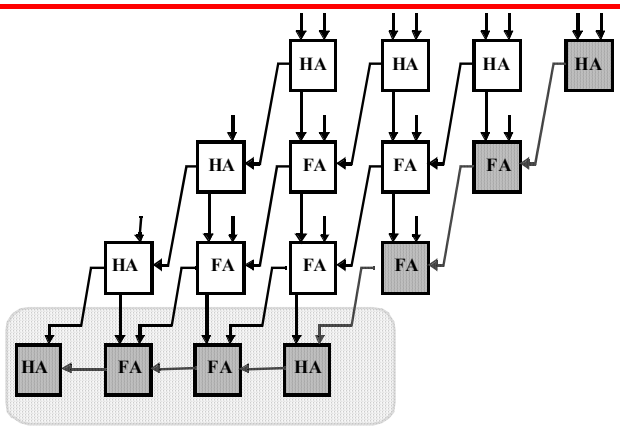


$t_{mult} = [(M-1)+(N-2)]t_{carry} + (N-1)\ t_{sum} + t_{and}$

Both carry and sum delays important: T-gate adder cell…

7

Carry-Save Multiplier



$t_{mult} = (N-1)t_{carry} + t_{and} + t_{merge}$

8

Booth Recoding

- To implement 2s complement multiplication, modified Booth recoding is typically used
- Idea: Recode the multiplier value in a higher radix in order to reduce the # of partial products
- Ex:

$$\begin{array}{ll} 010111 & (23) \\ \underbrace{011}_{1}\underbrace{11}_{3}\underbrace{10}_{2} & (30) \\ \hline & 690 \end{array}$$

9

Modified Booth Recoding

- Now we need to be able to multiply by 0,1,2, or 3
  - +3 is not easy to implement; requires two stages (+4X – 1X OR +2X + 1X)
- Instead look at three bits at a time and use negatives:
  - ±2X, ±1X, 0
  - Must be able to multiply by 0, 1, 2, -1, -2
  - 0 and 1 are easy, 2X involves a shift left by 1 bit position, -1X: invert all bits and set Cin = 1, -2X: invert all bits, carry in a 1, and shift left by 1 bit

10

Recoding table

- From MSB to LSB, look at multiplier bits two at a time along with MSB of next less-significant pair
- Instead of 3Y, use –Y, then increment next partial product to add 4Y
- Similarly, for 2Y, use –2Y, then increment next partial product to add 4Y

$x_{i+2}x_{i+1}x_i$	Add to partial product
000	+0Y
001	+1Y
010	+1Y
011	+2Y
100	-2Y
101	-1Y
110	-1Y
111	-0Y

11

Example

- 010111 (23)
  011110 (30)
  690

Originally: 011110

011 → +2

111 → 0

100 → -2

LSB extends with 0s

So we have:

(+2)(0)(-2)

12

Example, two 8-bit negative #'s

Extend sign in partial products

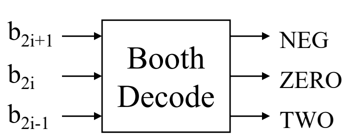
```
          10110010   = -78
X         10011101   = -99
--------------------------------
  1111111110110010
  00000001001110
  111101100100
  0010011100
--------------------------------
 10001111000101010   = 7722
```

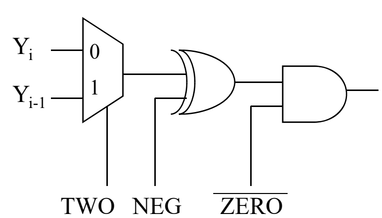
Ignore carry out into 17th place

Recode:
10011101

Booth Decoding and Partial Product Generation

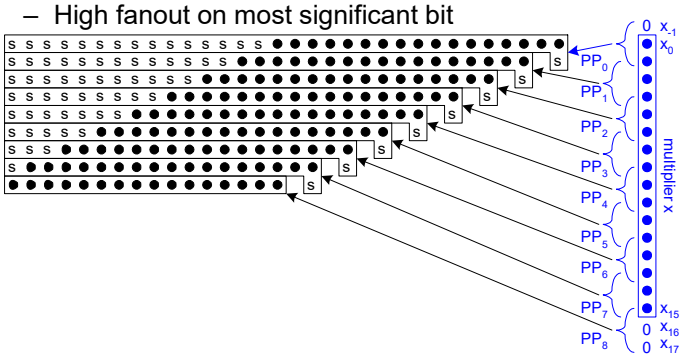
Operation	NEG	ZERO	TWO
x 0	0	1	0
x 1	0	0	0
x (-1)	1	0	0
x 2	0	0	1
x (-2)	1	0	1





Sign Extension

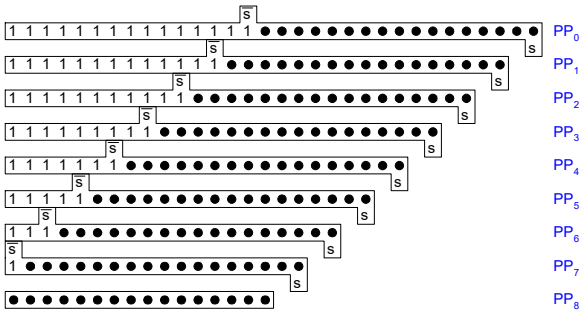
- Partial products can be negative
  - Require sign extension, which is cumbersome
  - High fanout on most significant bit



15

Simplified Sign Ext.

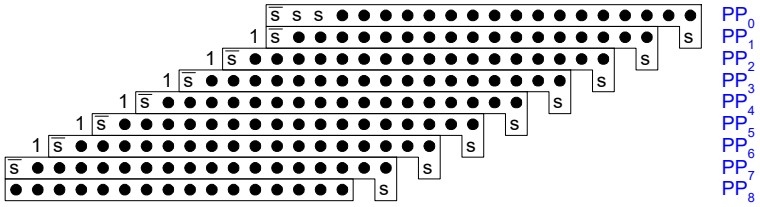
- Sign bits are either all 0's or all 1's
  - Note that all 0's is all 1's + 1 in proper column
  - Use this to reduce loading on MSB



16

Even Simpler Sign Ext.

- No need to add all the 1's in hardware
  - Precompute the answer!



17

Summary

- Generally, multiply function consists of AND functions to generate the partial products and lots of addition
  - Carry and sum delays of adder cells can be equally critical
- Modified Booth recoding reduces the # of partial products to be added, improves speed
  - Also suitable for 2s complement addition
- Other topics:
  - Can pipeline within the multiplier unit to improve throughput
  - Tree structures to reduce the # of adders needed and speed the result (speed becomes logarithmic in # of bits); see Fig 11.85, Wallace Tree

18